

Université de Montpellier II

TER master 1 informatique

Implémentation de protocole sur une
plateforme de réseaux de capteurs
sans-fils

H. Alatrasta, S. Aliaga, K. Gouaich, J. Mathieu
Encadrants : J. Champ, J.C. König, C. Saad

25 avril 2008

Remerciements

Nous tenons à remercier nos encadrants J. Champ, J.C. König et C. Saad pour toute l'aide apportée et pour nous avoir proposé ce sujet. Nous tenons aussi à remercier le laboratoire du lirmm de Montpellier pour nous avoir prêté le matériel nécessaire à nos expérimentations.

Table des matières

1	Présentation des Réseaux de capteurs sans-fil	5
2	TinyOs et NesC	8
2.1	Les capteurs	8
2.2	Le système d'exploitation TinyOS	10
2.2.1	Présentation	10
2.2.2	Propriétés	10
2.2.3	Allocation de la mémoire	11
2.2.4	Allocation des ressources :	11
2.3	Simulation : TOSSIM et TinyViz	12
2.4	NesC	14
2.4.1	Les Principales caractéristiques de NesC	14
2.4.2	Les fichiers dans NesC	14
2.4.3	Types de données	18
3	Les algorithmes AT-Free et AT-Dist	20
3.1	AT-Free	20
3.2	AT-Dist	21
3.3	Implémentation	23
4	Les tests	30
5	Conclusion	33
6	Annexes	34
6.1	Installation de Moteworks (windows)	34
6.1.1	Compilation	34
6.1.2	Chargeur : MoteConfig	34
6.1.3	XSniffer	37
6.2	Intallation et configuration sous linux	39
6.2.1	TinyOs 1	39
6.2.2	TinyOs 2	40
6.3	Comptes rendus des réunions	42
6.4	Fiche technique des capteurs micaz	44
7	Glossaire	47
8	Bibliographie	49

Table des figures

1.1	Un réseau de capteurs surveillant une forêt	6
2.1	Un capteur	8
2.2	Architecture d'un capteur	9
2.3	TinyViz	13
2.4	Architecture d'une application NesC	15
2.5	Processus de compilation	15
3.1	Approximation	21
3.2	Exemple de la méthode AT-Dist	22
3.3	Position estimée du nœud X	22
3.4	Représentation du réseau par une matrice	24
4.1	Topologie du test	31
4.2	Résultats du test	32
6.1	Le logiciel Moteconfig	36
6.2	Le logiciel XSniffer	38

Chapitre 1

Présentation des Réseaux de capteurs sans-fil

L'objectif du TER est d'implémenter sur des modules MICAz (marque Crossbow) des algorithmes de localisation dans un réseau de capteurs de type ad-hoc.

Les capteurs sont des objets électroniques capables de mesurer une grandeur physique (au sens large) : température, pression, pH, etc. Les modules MICAz sont des transmetteurs sans fil fonctionnant avec la technologie Zig-Bee, sorte de WiFi adapté aux appareils disposant de peu d'énergie. C'est le cas de ces modules car ils sont alimentés par piles. Ces transmetteurs sont prévus pour recueillir et transférer les mesures issues de capteurs sur lesquels ils se connectent. Pour ce TER, les données transmises sont uniquement liées aux algorithmes implémentés. Donc aucun véritable capteur n'est nécessaire. Cependant, dans la pratique, un capteur est indissociable d'un transmetteur. Voilà pourquoi nous ferons volontairement l'amalgame capteur - module par la suite.

Les réseaux de capteurs sont utilisés dans de nombreux domaines tels que l'industrie, le bâtiment (capteurs sismiques), l'écologie (contrôle des polluants, du climat et des désastres), le domaine médical ou bien encore le militaire (surveillance). C'est une technologie en plein essor et qui a de très nombreux débouchés. Par exemple, si une organisation forestière s'intéresse à la surveillance de son domaine forestier afin de détecter le départ d'un incendie, les capteurs dispersés dans le domaine s'avèrent très utiles pour faire remonter l'information jusqu'au point de contrôle (voir figure 1.1). Le capteur X diffuse un message d'alerte avec sa position qui est relayé de proche en proche par les capteurs jusqu'au point de collecte des informations où l'alerte peut être donnée. On estime le lieu de l'incendie par rapport à la position du capteur qui a lancé l'alerte. Comme les ressources d'un capteur sont très limitées, la réalisation d'une application dépend entièrement de la collaboration entre tous les capteurs qui exécutent des tâches simples.

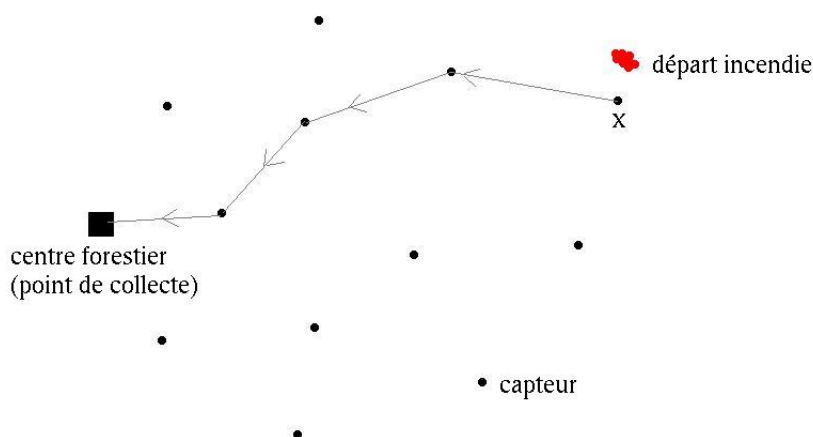


FIG. 1.1 – Un réseau de capteurs surveillant une forêt

Un réseau de type ad-hoc est un réseau sans hiérarchie entre les nœuds et où les informations transitent de nœud en nœud pour atteindre leur destination. Il est modélisé par un graphe (non orienté) où chaque nœud est ici un capteur et où chaque arête correspond à un lien de communication ZigBee. Pour les réseaux de capteurs, les mesures doivent atteindre un nœud particulier appelé point de collecte (le centre forestier de la figure 1.1). Autrement dit, de tout nœud du réseau, il doit exister un chemin vers le point de collecte.

La localisation permet de construire une cartographie complète du réseau à partir de la connaissance de la position de quelques nœuds du réseau. Ces nœuds particuliers portent le nom d'ancre. La cartographie complète d'un réseau de capteurs est nécessaire car une mesure représente l'état d'un point particulier. Or la localisation systématique, par GPS ou "manuellement", est une solution beaucoup trop coûteuse. Quand il y a un évènement on veut savoir où il s'est produit. De plus certains algorithmes de communication (routage, broadcast) utilisent aussi les positions.

Deux algorithmes de localisation, développés par C. Saad et al., sont à implémenter : AT-Free et AT-Dist (cf. [1] de la bibliographie). Les deux algorithmes sont des évolutions de la triangulation. AT-Free se contente de l'existence d'arête(s) autour d'un nœud pour tenter de localiser ce nœud. Le graphe correspondant n'est pas évalué. AT-Dist utilise en plus la puissance du signal pour valuer le graphe. La valuation est une distance déduite de la puissance, distance séparant le nœud courant au nœud à l'autre extrémité de l'arête.

Contexte et contraintes : Les modules MICAz sont programmables, dans des conditions particulières comme souvent en informatique embarquée. La programmation ne peut pas être effectuée directement sur le module. Ils

sont donc interfacés au PC via USB grâce à un autre module sur lequel ils s'enchâssent : le MIB 520 (Crossbow). Les modules MICAz sont dotés d'un système d'exploitation, appelé TinyOS, adapté au contexte "capteur" c'est-à-dire à un environnement industriel, à des faibles capacités de traitement et à une faible énergie embarquée. Avec TinyOS, il n'existe qu'un unique langage de programmation de haut niveau : NesC. Il s'agit d'une variante du langage C adaptée à ce système d'exploitation et donc aux capteurs. Les outils de programmation (compilateur, chargeur, simulateur, etc) sont bien entendu spécifiques. L'outil fourni avec les modules s'appelle MoteWorks. Il existe également des outils libres, sous forme d'une distribution Ubuntu dédiée : XubuntuOS.

Chapitre 2

TinyOs et NesC

2.1 Les capteurs

Les capteurs MICAZ sont de taille très réduite et sont limités en ressource. Ils sont autonomes et capables de traiter des informations ainsi que de les transmettre via des ondes radio (ZigBee). Trois types de fréquences sont possibles pour le signal ZigBee :

- 868-868.6 MHz (Europe)
- 902-928 MHz (Amérique du nord)
- 2400-2483.5 MHz (International)

Un capteur analyse son environnement (il effectue des relevés de mesures) et transmet les informations aux autres capteurs se trouvant dans sa zone de couverture. L'information se propage de capteur en capteur, traversant plusieurs zones ; le réseau se trouve ainsi complètement couvert.

Sur la figure 2.1 un modèle de capteur :

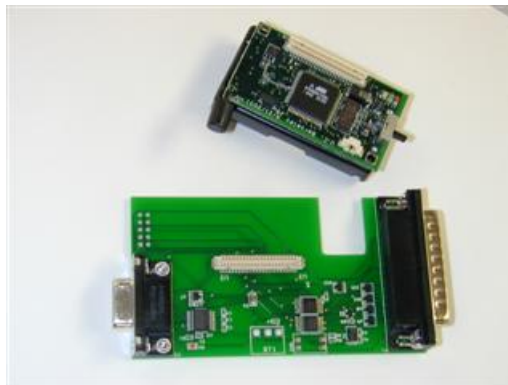


FIG. 2.1 – Un capteur

Architecture générique d'un Mote

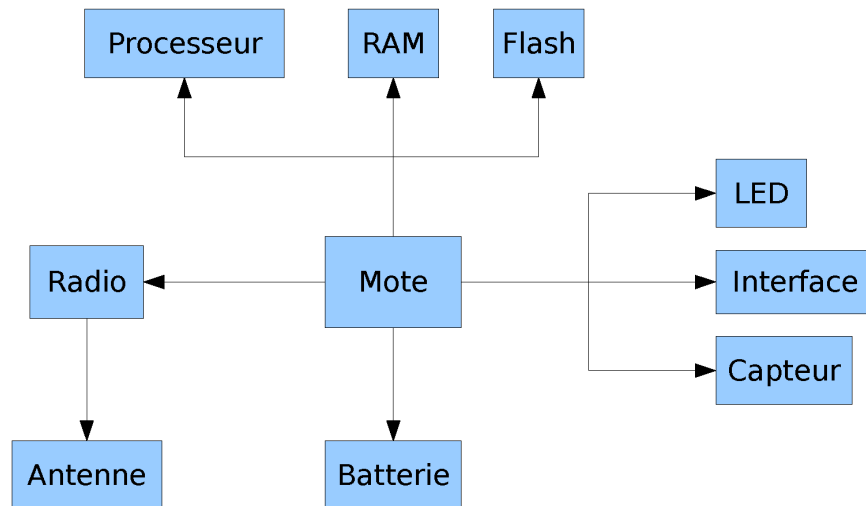


FIG. 2.2 – Architecture d'un capteur

Architecture d'un capteur

L'architecture des capteurs, en général, est commune pour tous. On peut voir sur la figure 2.2 les différents composants qui constituent un capteur.

Mote, Processeur, RAM, Flash :

On appelle Mote la carte physique utilisant le système d'exploitation. Le processeur est à la base des calculs binaires. Les mémoires RAM et Flash servent pour le stockage, définitif ou non, des données.

Radio, Antenne :

Afin d'émettre, un capteur a besoin d'une antenne et d'une radio pour ajuster les fréquences hertziennes.

LED, interfaces, capteur :

Composants prévus pour mettre en place un réseau de capteurs.

Batterie :

Indispensable pour l'autonomie du capteur.

2.2 Le système d'exploitation TinyOS

2.2.1 Présentation

TinyOS est le système d'exploitation open-source pour les réseaux de capteurs sans-fil conçu par l'université américaine de BERKELEY. Sa conception a été entièrement réalisée en NesC, langage orienté composant qui se rapproche syntaxiquement du langage le plus connu : le C.

TinyOS a été créé pour répondre aux caractéristiques et aux nécessités des réseaux de capteurs, telles que :

- Une taille de mémoire réduite.
- Une basse consommation d'énergie.
- Des opérations d'assistance intensive.
- Des opérations robustes.
- Il est optimisé en termes d'usage de mémoire et d'énergie.

2.2.2 Propriétés

Le plus gros avantage de TinyOS est qu'il est basé sur un fonctionnement événementiel, c'est-à-dire qu'il ne devient actif qu'à l'apparition de certains événements. Le reste du temps, le capteur se trouve en état de veille afin de garantir une durée de vie maximale aux faibles ressources énergétiques du capteur. TinyOS se distingue aussi par son caractère non préemptif, c'est-à-dire qu'il ne gère pas les interruptions entre tâches. Par contre il donne une priorité aux interruptions matérielles qui peuvent à tout moment stopper l'exécution d'une tâche. Pour terminer, TinyOS ne gère pas de "temps réel" car il n'est pas prévu pour manipuler des niveaux de priorité, pour mieux respecter les échéances, dans les tâches. TinyOS est donc basé sur une structure à deux niveaux de planification :

- Les événements : ils sont utilisés pour réaliser de petits processus (par exemple quand le compteur du «timer» arrive à son terme). De plus ils peuvent interrompre les tâches qui sont exécutées.
- Les tâches : les tâches sont pensées pour réaliser une plus grande quantité de traitements et elles ne sont pas critiques dans le temps. Les tâches sont exécutées complètement, mais l'initialisation et la terminaison d'une

tâche sont des fonctions séparées.

2.2.3 Allocation de la mémoire

Il est très important d'aborder la façon avec laquelle un système d'exploitation gère la mémoire et plus spécialement quand celui-ci travaille dans un espace restreint. TinyOS ne nécessite pas beaucoup de place mémoire puisqu'il n'a besoin que de 300 à 400 octets dans le cadre d'une distribution minimale. Il est primordial d'avoir 4 Ko de mémoire libre qui se répartissent entre les différents besoins suivant :

La Pile

Elle sert de mémoire temporaire pour l'empilement et le dépilement des variables locales.

Les variables globales

Elles réservent un espace mémoire pour stocker des valeurs pouvant être accessibles depuis différentes tâches.

La mémoire libre

Pour tout le reste du stockage temporaire. La notion d'allocation dynamique de mémoire n'est pas présente dans le système, ce qui simplifie l'implémentation mais, par ailleurs, il n'existe pas de mécanisme de protection de la mémoire, ce qui rend le système plus vulnérable au crash et aux corruptions de mémoire.

2.2.4 Allocation des ressources :

L'ordonnanceur

Le choix d'un ordonnanceur détermine le fonctionnement global du système et le dote de propriétés telles que la capacité à fonctionner en temps réel. L'ordonnanceur TinyOS se compose de :

- 2 niveaux de priorités (bas pour les tâches, haut pour les événements).
- 1 file d'attente FIFO (disposant d'une capacité de 7).

On a un niveau de priorité entre les tâches leur permettant de se classer. Lors de l'arrivée d'une nouvelle tâche, celle-ci sera placée dans la file d'attente en fonction de sa priorité. Dans le cas où la file d'attente est pleine, la tâche dont la priorité est la plus faible est enlevée de la file FIFO.

Les tâches

Elles sont utilisées pour effectuer la plupart des blocs d'instructions d'une application. A l'appel d'une tâche, celle-ci va prendre place dans une file d'attente de type FIFO (First In First Out) pour y être exécutée. Une tâche activée s'exécute entièrement car il n'y a pas de mécanisme de préemption. Lorsque la file est vide, le système met en veille le dispositif jusqu'au lancement de la prochaine interruption.

Les évènements

Ils sont prioritaires par rapport aux tâches et peuvent interrompre la tâche en cours d'exécution. Ils permettent de faire le lien avec les interruptions matérielles.

Plates-formes sous TinyOS

TinyOS est prévu pour fonctionner sous plusieurs plates-formes comme Windows (2000 et XP) ou bien GNU/Linux. Deux principales versions de TinyOS sont disponibles : la version stable (V. 1.1.15) et La version en développement (V. 2.0.2) qui nécessite l'installation de l'ancienne version pour fonctionner.

2.3 Simulation : TOSSIM et TinyViz

Le simulateur TOSSIM

Afin de simuler le comportement des capteurs, un outil très puissant a été développé et proposé sous le nom de TOSSIM. Ce dernier est souvent utilisé avec une interface graphique (TinyViz) pour une meilleure compréhension et visualisation de l'état du réseau. L'utilisation de ces deux logiciels est immédiate dès lors que TinyOS est opérationnel.

TinyViz

L'outil TinyViz est une application graphique qui nous permet d'avoir un aperçu de notre réseau sans avoir à déployer les capteurs dans la nature. Une économie d'effort et une préservation du matériel sont possibles grâce à cet outil. L'application permet une analyse étape par étape en activant les différents modes disponibles. Comme nous le voyons sur la figure 2.3. Les capteurs sont disposés dans la partie gauche de la fenêtre du logiciel.

Mode d'emploi

Dans cette section nous allons détailler un peu ce que fait chaque bouton présent dans l'interface :

On/Off : il met en marche ou éteint un capteur.

Delay : il permet de sélectionner la durée au bout de la quelle se déclenche le timer.

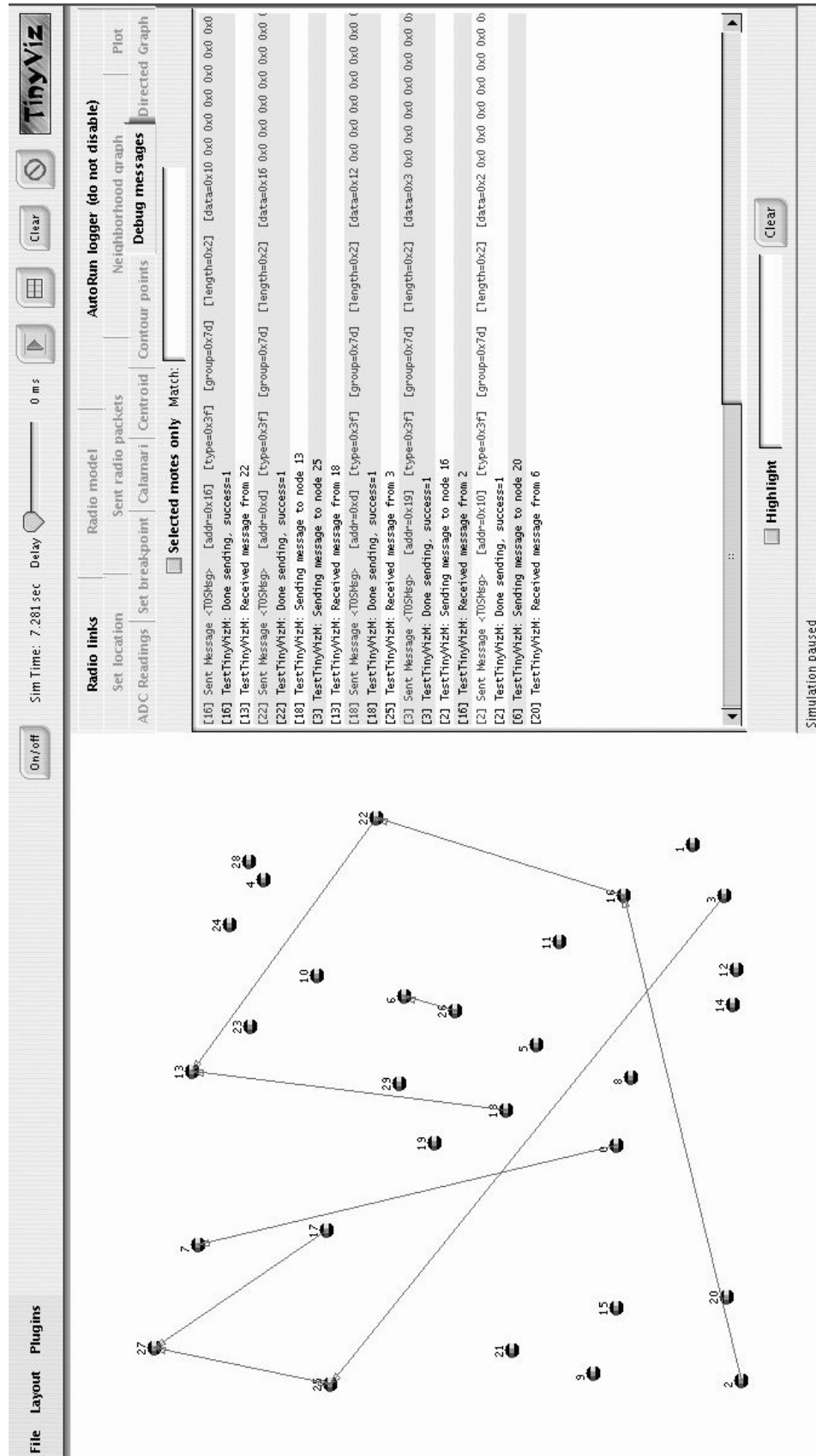


FIG. 2.3 – TinyViz

Play : il permet de lancer la simulation ou de la mettre en pause.

Grilles : il permet d'avoir une grille pour situer les capteurs dans l'espace.

Clear : il efface tous les messages qui transitent entre les capteurs.

Arrêt : il met fin à la simulation.

Plusieurs plugins sont disponibles pour visualiser la simulation sous différentes formes. Les plugins dont on s'est beaucoup servi, sont "debug message" pour afficher tous les messages de type `dbg(**,**,**)` afin de vérifier la nature des messages et "radio links" qui nous permet de voir, avec des flèches ou des cercles, si un capteur est en train d'émettre ou non.

2.4 NesC

Le langage NesC (network embedded system C) est un dialecte de C basé sur des composants. NesC est orienté pour satisfaire les exigences des systèmes embarqués. De plus, il supporte un modèle de programmation qui agrège l'administration des communications, les concurrences provoquant les tâches et les événements ainsi que la capacité de réagir par rapport à ces événements.

NesC réalise aussi une optimisation dans la compilation du programme, en détectant les carrières possibles de données qui peuvent produire des modifications concurrentes au même état, à l'intérieur du processus d'exécution de l'application. Une carrière de données se produit quand plus d'un fils peuvent simultanément accéder à la même section de mémoire (concurrence d'accès mémoire entre threads), et quand au moins l'un des accès est un "write". NesC simplifie aussi le développement d'applications et réduit la taille du code.

2.4.1 Les Principales caractéristiques de NesC

NesC est constitué d'interfaces et de composants.

Une interface peut être utilisée ou peut être fournie. Les composants sont des modules ou des configurations.

Une application est représentée comme un ensemble de composants, regroupés et rattachés entre eux (voir figure 2.4).

Les interfaces sont utilisées pour les opérations qui décrivent l'interaction bidirectionnelle. Le fournisseur de l'interface doit mettre en application des commandes, alors que l'utilisateur de l'interface doit mettre en application des événements.

Deux types de composants existent :

Les modules, qui mettent en application des spécifications d'un composant.

Les configurations, qui se chargeront d'unir différents composants en fonction des interfaces (commandes ou événements).

La figure suivante montre un diagramme de blocs dans lequel est décrit le processus de compilation pour une application TinyOS écrite en NesC :

2.4.2 Les fichiers dans NesC

Les fichiers de NesC sont classés en trois types : Interfaces, modules et configurations.

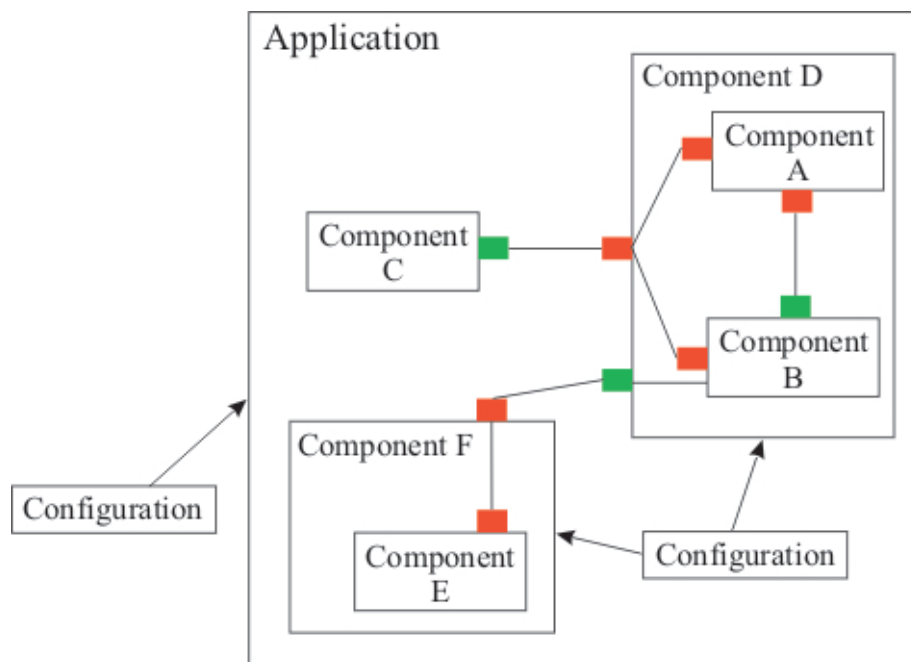


FIG. 2.4 – Architecture d'une application NesC

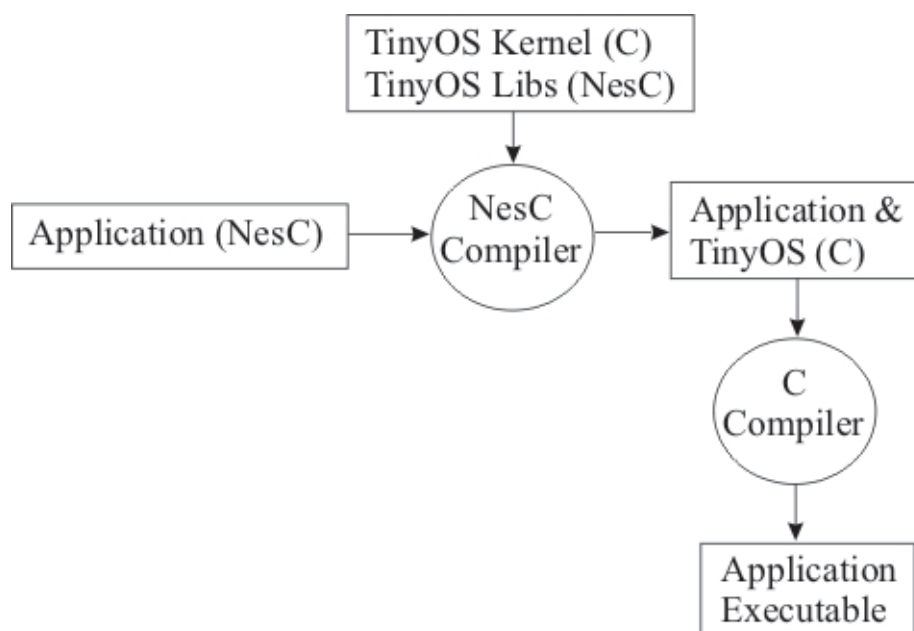


FIG. 2.5 – Processus de compilation

Interface :

Ce type de fichier déclare les services fournis et les services qui seront utilisés. Ils se trouvent dans le répertoire /tos/interface. (exemple : StdControl.nc).

Module :

Le type Module contient le code de l'application, en mettant en œuvre une ou plusieurs interfaces. (exemple : BlinkM.nc)

Configuration :

Dans ces fichiers on déclare la manière d'unir les différents composants et comment effectuer le contrôle des flux. (exemple : Blink.nc).

Composants

TinyOS définit un nombre important de concepts qui sont exprimés dans NesC. D'abord, les applications NesC sont construites par des composants avec des interfaces bidirectionnelles définies. Aussi, NesC définit un modèle basé sur les tâches et les captures d'événements matériels, et détecte des éclatements d'information pendant la compilation.

Un composant, du point de vue de la programmation, est composé de plusieurs sections et l'ensemble de toutes ces sections donne lieu à la création de ce composant.

Implémentations

Cette section définit les connections entre les différents composants qu'utilise l'application. Dans cette section «implementation» sont donc principalement définis quels sont les composants qui fournissent les interfaces à notre application (ils seront généralement des composants primitifs). Généralement nous devons utiliser les interfaces que nous fournissent d'autres composants primitifs ou non primitifs, et en définitive pour chacune de ces interfaces, que nous utiliserons dans la création de notre composant, on doit obligatoirement définir des relations avec les composants qui fournissent ces interfaces. Le processus définissant ces relations s'appelle «wiring».

Exemple :

```
1 implementation {  
2   components Main, MonAppliM ;  
3   Main.StdControl -> MonAppliM.StdControl;  
4 }
```

Configurations

C'est à cet endroit que l'on déclare les autres composants dont se servira l'application. Cette possibilité offerte par le langage permet de faire de la programmation modulaire et de réutiliser des composants préalablement définis.

La structure de la partie Configurations est la même que celle de la partie Implémentation.

Exemple :

```
1 Configuration MonAppli {
2   implementation {
3     components GenericComm, TimerC;
4     TimerC.StdControl -> GenericComm.StdControl;
5   }
6 }
```

Module

Cette partie du code est généralement plus étendue et c'est dans celle-ci que l'on programme réellement le comportement qu'on souhaite voir réalisé par l'application. Cette partie là est à son tour divisée en trois sous-sections : Uses, Provides, Implementation.

Exemple :

```
1 module MonAppliM {
2   provides {... }
3   uses {... }
4 }
5 implementation {... }
```

La première sous-section, «provides», indique au compilateur les interfaces que va fournir notre composant. Par exemple, si notre composant est une application, on doit fournir au moins l'interface StdControl.

```
1 provides {
2   interface interfaceque nous fournissons;
3 }
```

La sous-section «uses» informe le compilateur que nous allons faire usage d'une interface (on pourra donc effectuer des appels aux méthodes de cette interface). Pour faire cela, on a besoin de respecter quelques règles : si nous utilisons une interface, il nous faut avoir dans la section «implementation» un lien ("wiring") reliant cette interface avec un composant qui la fournit. Finalement, utiliser une interface oblige implicitement à gérer les événements pouvant se produire du fait d'avoir utilisé cette interface précise.

```
1 uses {
2   interface interfaceque nous utilisons;
3 }
```

La sous-section «implementation», est celle qui contiendra toutes les méthodes nécessaires pour fournir le comportement souhaité à notre composant ou à notre application. Cette sous-section doit contenir au moins :

- Les variables globales que va utiliser notre application
- Les fonctions qu'elle doit mettre en œuvre pour les interfaces qui nous fournissons
- Les événements qu'elle doit mettre en œuvre venant des interfaces que nous utilisons.

2.4.3 Types de données

Les types de données qui peuvent être utilisés en NesC sont tous ceux que fournit le langage C standard plus quelques autres qui n'apportent pas de puissance de calcul mais qui sont très utiles pour la construction de paquets puisqu'ils fournissent à l'utilisateur le nombre de bits qu'ils occupent (ceci est important au moment de la transmission des informations par l'intermédiaire des ondes radio). Ces types additionnels sont :

- `uint16_t` : entier non signé sur 16 bits.
- `uint8_t` : entier non signé sur 8 bits.
- `result_t` : utilisé pour savoir si une fonction a été exécuté avec succès ou non, c'est comme un booléen mais avec les valeurs SUCCESS et FAIL. (retour de fonction)
- `bool` : valeur booléenne qui peut être TRUE ou FALSE.

En NesC il est possible de faire une utilisation dynamique de la mémoire mais ce n'est pas très recommandé (à moins que cela ne soit absolument nécessaire). Pour pouvoir l'utiliser il existe un composant spécial appelé MemAlloc qui permet une gestion dynamique de la mémoire.

Types de fonctions en NesC

En NesC les fonctions peuvent être de types très variés. Il y a d'abord les fonctions classiques avec la même sémantique qu'en C et la façon de les invoquer est aussi la même qu'en C. Il y a aussi des types supplémentaires de fonctions : `task`, `event`, `command`. Les fonctions «command» sont principalement des fonctions qui sont exécutées de manière synchrone, c'est-à-dire que lorsque elles sont appelées elles sont exécutés immédiatement. La manière d'appeler ce genre de fonction est :

```
1 call interface.nomFonction
```

Les fonctions «task» sont des fonctions qui sont exécutées dans l'application, utilisant la même philosophie que les fils ou «threads», c'est principalement une fonction normale qui est invoquée de la manière suivante :

```
1 post interface.nomTache
```

Immédiatement après son invocation, l'exécution du programme qui l'a invoqué se poursuit.

Les fonctions «event» sont des fonctions qui sont appelées quand on relèvera un signal dans le système, elles possèdent principalement la même philosophie que la programmation orientée événements, de sorte que, lorsque le composant reçoit un événement, on effectue l'invocation de cette fonction. Il existe une méthode pour pouvoir invoquer manuellement ce type de fonctions :

```
1 signal interface.nomEvenement
```

Différences entre NesC et C, C++ et Java

Les langages C, C++ et Java ont une composition dynamique et ont un espace de noms (namespace) global. Appeler une fonction requiert d'utiliser un nom qui est unique pour cette fonction dans l'espace de noms. NesC a un autre objectif. D'abord, le code est divisé en composants, unités discrètes de fonctionnalités. Un composant peut seulement faire référence à des variables de son propre espace de noms. D'une certaine manière, les composants NesC sont semblables à des objets. La principale différence est la portée des appels par référence. Tandis que les objets C++ et Java font référence à des fonctions et des variables au niveau global, les composants NesC utilisent uniquement un niveau local. Ceci signifie que, non seulement il faut déclarer les fonctions mises en œuvre, mais que de plus un composant doit déclarer les fonctions qui appellent ces fonctions.

Conclusions

NesC est un langage de programmation qui présente de grands avantages pour le développement d'applications pour des systèmes embarqués, et particulièrement pour les réseaux de capteurs sans fil. Celui-ci apporte des outils pour une bonne programmation d'applications, d'abord en abstrayant les caractéristiques de modularité que présentent les réseaux de capteurs, et ensuite en facilitant la mise en œuvre de programmes pour ces systèmes.

Chapitre 3

Les algorithmes AT-Free et AT-Dist

AT-Free et AT-Dist sont deux algorithmes de localisation par approximation. Leur but est de permettre aux capteurs de trouver leur position à l'aide des positions connues de seulement quelques capteurs spécifiques appelés "ancres".

3.1 AT-Free

Techniques d'approximation

Pour calculer la position estimée (définie comme un point de coordonnées (Ex, Ey) d'un capteur appelé "node" on se sert de la position d'autres capteurs (les ancres) dont la position est déjà connue.

Pour calculer cette position estimée (qui est le centre de gravité de la zone où le node se trouve) on a 3 techniques. L'utilisation d'une technique plutôt que d'une autre dépendra des caractéristiques des capteurs. Par exemple, pour la technique utilisant la puissance du signal (RSSI), il faut que les capteurs soient équipés du module de traitement RSSI.

Technique d'approximation AT-Free

Avec cette technique, un capteur cherche à calculer sa position à partir des informations reçues sur le réseau.

Chaque ancre diffuse sa position (x,y) . Chaque capteur recevant le message le diffuse à son tour.

Quand un capteur reçoit le message, il déduit le nombre de sauts entre lui même et l'ancre d'après la portée du signal. Si le nombre de sauts est de un (i.e. l'ancre et le capteur sont voisins) le capteur se trouve dans le cercle de rayon r (la portée du signal) et centré sur la position de l'ancre. Par contre, si le capteur et l'ancre ne sont pas voisins alors le capteur se trouve dans le cercle de rayon $r \times h$ (h : nombre de sauts effectués par le message) et centré sur l'ancre.

Lorsque le capteur reçoit la position d'une seconde ancre il sait qu'il se trouve

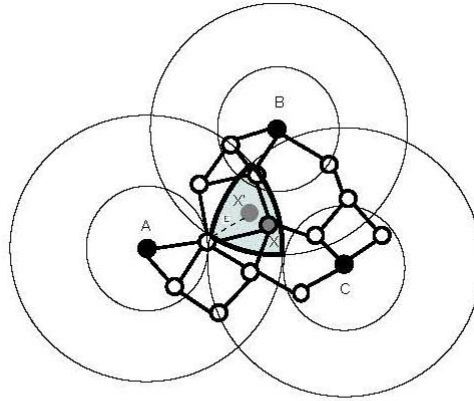


Fig : Approximation

FIG. 3.1 – Approximation

dans l'intersection des deux cercles. Il prend comme coordonnées estimées le centre de gravité de cette zone, avec une erreur epsilon égale à la distance entre ce point estimé et le point le plus éloigné de la zone où il se trouve.

Quand le capteur a assez bien estimé sa position (l'erreur epsilon est passée en dessous d'un certain seuil) on le considère comme une ancre et il commence à diffuser sa propre position.

Remarque : Pour avoir un résultat cohérent on a besoin de recevoir les informations d'au moins 2 ancres.

3.2 AT-Dist

AT-Dist fonctionne comme AT-Free mais utilise en plus la distance estimée que parcourent les messages pour affiner la recherche de position. Cet algorithme utilise RSSI : plus loin est la source du message, moins le signal reçu sera fort.

Cette méthode d'approximation des distances est nommée Sum-Dist. Il suffit de sommer les distances intermédiaires entre deux points (en nombre de sauts). Voir l'exemple de la figure 3.2



FIG. 3.3 – Position estimée du nœud X

3.3 Implémentation

L'algorithme présenté par nos tuteurs est implémenté de façon à optimiser les temps de calcul car la puissance de calcul est une ressource critique dans le cas des systèmes embarqués. Dans notre cas, nous devons simuler une topologie d'une cinquantaine de capteurs, en prenant en compte les temps de transfert des messages ainsi que la capacité de la file où sont stockés les messages, qui est limitée à 7. Le but de cet algorithme est de calculer les positions estimées (E_x, E_y) ainsi que le taux d'erreur (ψ), des nœuds, en fonction des coordonnées qu'on reçoit (x_O, y_O) et du nombre de sauts ($r2 = \text{Nombre de sauts} * \text{portée du rayon}$).

Le code de la fonction principale :

```
1 void faireDisque(uint8_t x0, uint8_t y0, uint8_t
   r1, uint8_t r2) {
2     uint32_t aX, aY;
3     uint8_t nb2;
4
5     float epsi = 0.0;
6     float d = 0.0;
7     float d2 = 0.0;
8     uint8_t liminfx0, limsupx0;
9     uint8_t liminfy0, limsupy0;
10
11     if (x0 < r2+1) liminfx0=0; else liminfx0 = x0-r2
        -1;
12     limsupx0 = x0+r2+1; if (limsupx0 > 50) limsupx0=
        MAXI-1;
13     if (y0 < r2+1) liminfy0=0; else liminfy0 = y0-r2
        -1;
14     limsupy0 = y0+r2+1; if (limsupy0 > 50) limsupy0=
        MAXI-1;
15     aX=0; aY=0; nb2=0; //Px=0; Py=0;
16     Matriz[x0][y0] += 1; Matriz[x0][y0] += 1;
17     for (i=liminfx0; i < limsupx0+1; i++) {
18         for (j=liminfy0; j < limsupy0+1; j++) {
19             d = distance(i, j, x0, y0);
20             if ((d > r1) && (d <= r2)) {
21                 Matriz[i][j] += 1;
22                 if (vMAXI < Matriz[i][j]) {
23                     vMAXI = Matriz[i][j];
24                 }
25             }
26         }
27     }
```

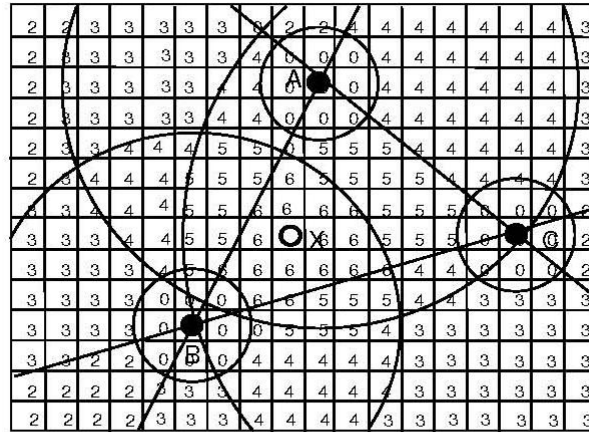


Fig : Représentation du réseau par une matrice

FIG. 3.4 – Représentation du réseau par une matrice

```

28  if (compt>1) {
29      for (i=liminfx0; i<limsupx0+1; i++) {
30          for (j=liminfy0; j<limsupy0+1; j++) {
31              if(Matriz[i][j] == vMAXI) {
32                  aX+=i;aY+=j;
33              nb2++;
34              }
35          }
36      }
37      Ex=(aX/(float)nb2); Ey=aY/(float)nb2;
38      for (i=liminfx0; i<limsupx0+1; i++) {
39          for (j=liminfy0; j<limsupy0+1; j++) {
40              if(Matriz[i][j] == vMAXI) {
41                  d2 = distance(i,j,Ex,Ey);
42                  if(eps < d2) eps = d2;
43              }
44          }
45      }
46      psi = (uint8_t)eps;
47  }

```

Une deuxième fonction "defaireDisque(xO,yO,r1,r2)" a aussi été implémentée afin de prendre en compte le cas d'une meilleure estimation, soit en nombre de sauts soit en taux d'erreur. Le but est de décrémenter la valeur (vMAXI) qui représente l'intersection entre les différentes surfaces (la zone où se trouve le capteur).

Justification : Admettons une topologie à deux nœuds et une ancre. Soit le nœud N1 atteignable directement par l'ancre A, donc N1 devrait calculer sa position par rapport aux coordonnées de A avec $r2 = 1 * PORTEE$. Si entre temps N1 reçoit un message, type relais, de la part de N2, ce dernier devrait communiquer les coordonnées de A mais avec un nombre de sauts $\neq 1$, exactement 2. Dans ce cas, N1 pourrait être ramené à calculer sa position selon les coordonnées de A mais avec un nombre de sauts supérieurs.

```

1 void defaireDisque(uint8_t x0, uint8_t y0,
2   uint8_t r1, uint8_t r2) {
3   float d = 0.0;
4   uint8_t liminfx0, limsupx0;
5   uint8_t liminfy0, limsupy0;
6
7   if (x0<r2+1) liminfx0=0; else liminfx0 = x0-r2
8     -1;
9   limsupx0 = x0+r2+1; if (limsupx0>50) limsupx0=
10    MAXI-1;
11   if (y0<r2+1) liminfy0=0; else liminfy0 = y0-r2
12     -1;
13   limsupy0 = y0+r2+1; if (limsupy0>50) limsupy0=
14    MAXI-1;
15   Matriz[x0][y0] -=1;
16   for (i=liminfx0; i<limsupx0+1; i++) {
17     for (j=liminfy0; j<limsupy0+1; j++) {
18       d = distance(i, j, x0, y0);
19       if ((d>r1) && (d<=r2)) {
20         if (Matriz[i][j]==vMAXI) vMAXI -=1;
21         Matriz[i][j] -=1;
22       }
23     }
24   }
25 }

```

Communication

Afin de permettre un flux de communication entre les capteurs, notre implémentation se base essentiellement sur la gestion des évènements "réception" et "émission" en respectant le standard de NesC. Une consultation de la fiche technique et des divers exemples fournis avec la distribution de tinyOS est plus que nécessaire afin de mettre correctement en place un système de transmission.

Structure des messages

Nous nous sommes mis d'accord sur un type de message représentant toutes les données nécessaires pour la localisation des nœuds. Le message est

représenté par une structure de données définie dans le fichier d'entête "loc.h" comme ceci :

```
1 typedef struct {
2     int8_t id_ancre; // l'identifiant du capteur (
        ancre ou noeud)
3     int8_t x; // coordonnee cartesienne sur l'axe
        des abscisses
4     int8_t y; // coordonnee cartesienne sur l'axe
        des ordonnees
5     int8_t z; // troisieme coordonnee pour une
        representation en 3D.
6     int8_t epsilon; // le taux d'erreur
7     int8_t nb_sauts; // le nombre de sauts
8     int8_t id_msg; // l'identifiant du message (
        permet de tracer le message)
9 } msg_loc ;
```

Envoi de messages

La fonction d'émission *SendData()* est caractérisée par le fait que c'est une tâche et non pas un évènement et par conséquent elle doit être appelée avec l'instruction standard *post call SendData()* et implémentée avec le mot clé *void task*.

```
1 void task SendData() {
2     if (sending_packet) return;
3     atomic sending_packet = TRUE;
4     msgE.data[0] = loc1.id_ancre;
5     msgE.data[1] = loc1.x;
6     msgE.data[2] = loc1.y;
7     msgE.data[3] = loc1.z;
8     msgE.data[4] = loc1.epsilon;
9     msgE.data[5] = loc1.nb_sauts;
10    msgE.data[6] = loc1.id_msg;
11
12    if (call SendMsg.send(TOS_BCAST_ADDR, sizeof(
        msg_loc), &msgE) != SUCCESS)
13        { sending_packet = FALSE; return; }
14    call Leds.greenToggle(); //diode verte clignote
15    return;
16 }
```

On remarquera dans le corps de la fonction qu'on fait appel à la primitive *SendMsg.send()* qui se charge d'envoyer le message. Le standard NesC fournit aussi une autre primitive *SendMsg.sendDone()* qui se définit comme étant un évènement indiquant si une transmission a bien eu lieu ou pas. Les deux primitives sont fortement liées et, par conséquent, elles doivent être implémentées

en même temps.

Le code de la primitive *SendMsg.sendDone()* :

```
1  event result_t SendMsg.sendDone(TOS_MsgPtr msg,
   result_t success) {
2      atomic sending_packet = FALSE;
3      return success;
4  }
```

Réception de messages

La fonction de réception *ReceiveMsg.receive()* est un évènement et donc doit être implémentée avec le mot clé *event* :

```
1  event TOS_MsgPtr ReceiveMsg.receive(TOS_MsgPtr
   msg) {
2      TOS_Msg msg_tmp = *msg; // tampon pour eviter
   ecrasement du msg
3      loc2 = *(msg_loc *)msg_tmp.data;
4      test = 0;
5      sonde = FALSE;
6      if (loc2.id_msg == 100) {
7          sonde = TRUE;
8          if (loc2.id_ancre==TOS_LOCAL_ADDRESS) {
9              loc1.id_ancre = TOS_LOCAL_ADDRESS;
10             loc1.x = Ex;
11             loc1.y = Ey;
12             loc1.z = 100;
13             loc1.epsilon = psi;
14             loc1.nb_sauts = 0;
15             loc1.id_msg = 100;
16             post SendData();
17         } }
18         if (sonde == FALSE) {
19             loc2.nb_sauts ++;
20             loc2.id_msg = TOS_LOCAL_ADDRESS;
21             if ((loc2.id_ancre!=TOS_LOCAL_ADDRESS)&&(loc2
   .nb_sauts < seuil))
22                 {loc1=loc2; post SendData();} // limitation
   de la diffusion
23             if (localisee == FALSE) { // on entre ici que
   si localisation en cours
24                 for(i=0;i<mote;i++) {
25                     if (loca[i].id_ancre == loc2.id_ancre) {
26                         test = 1;
27                         defaireDisque(loca[i].x,loca[i].y,0,loca
   [i].nb_sauts*PORTEE);
```

```

28         if (loca[i].nb_sauts > loc2.nb_sauts)
                loca[i]=loc2;
29         else if ((loca[i].nb_sauts == loc2.
                    nb_sauts) && (loca[i].epsilon > loc2.
                    epsilon)) loca[i]=loc2;
30         else if ((loca[i].id_ancre == -1)&&(test
                    ==0)) { test = 1; loca[i]=loc2; loc2.z=i
                    ;}
31         if (test == 1) { // dans ce cas, mise a
                            jour effectuee
32             if (compt<mote) compt++;
33             // si epsilon local > seuil2 =>
                            estimation encore necessaire
34             if (psi>=rho) {
35                 call Leds.yellowToggle();
36                 faireDisque(loc2.x, loc2.y, 0,loc2.
                            nb_sauts*PORTEE);
37             }
38             // si epsilon local < seuil1 => ancre
                            estimee moyennement bonne
39             if (psi < psi2) {
40                 //while((psi<psi2)&&(psi2>rho))
41                     //{psi2=psi2*delta;}
42                 oc2.id_ancre = TOS_LOCAL_ADDRESS;
43                 loc2.x = Ex;
44                 loc2.y = Ey;
45                 loc2.epsilon = psi;
46                 loc2.nb_sauts = 0;
47                 loc1=loc2;
48                 post SendData();
49                 call Leds.greenToggle();
50             }
51             // si epsilon < rho => ancre estimee
                            bonne => diffusion (1 fois)
52             if (psi<=rho) {
53                 call Leds.redOn();
54                 localisee = TRUE;
55                 loc2.id_ancre = TOS_LOCAL_ADDRESS;
56                 loc2.x = Ex;loc2.y = Ey;
57                 loc2.epsilon = psi;loc2.nb_sauts = 0;
58                 loc1=loc2;
59                 post SendData();
60             } }
61         } }
62     return msg; }

```

La sonde Nous avons réalisé un autre type de capteur, qui nous sert de sonde. La sonde est activée après une ou deux minutes, lorsque l'algorithme s'est stabilisé dans le réseau, elle se charge d'interroger les nœuds, avec un type de message précis. Les nœuds lui répondent en envoyant leur position estimée que nous récupérons avec le sniffer. (sinon nous n'avons aucun moyen de connaître les résultats auxquels sont arrivés les capteurs). On se promène donc dans le réseau de capteurs avec la sonde et le sniffer pour recueillir les positions estimées de chaque capteur.

Chapitre 4

Les tests

Au moment de l'écriture de ce rapport nous n'avons malheureusement fait que quelques tests, avec peu de capteurs, dont un exemple est donné ici. En accord avec nos encadrants, des tests plus complets seront fournis le jour de la soutenance.

Conditions du test

La figure 4.1 donne la topologie que nous avons utilisée.

- portée = 10
- seuil de diffusion = 3
- $\psi_2 = \gamma = 7$
- $\rho = 3$

Résultats du test

Voir la capture d'écran du sniffer (placé entre le nœud 1 et le nœud 2) sur la figure 4.2

Le nœud 1 trouve (17,13) avec $\epsilon = 4$

Le nœud 2 trouve (17,21) avec $\epsilon = 4$

Analyse du test

Après réception des messages des ancres 10 et 11 les noeuds 1 et 2 se positionnent en (17,10) et (17,12) avec des epsilons de 7 et 16. Les noeuds ne diffusent pas leur position à cause d'epsilon trop grand. Ils faut donc avoir recours à la sonde (lignes 5 à 12).

Le noeud 1 se positionne donc au milieu des ancres 10 et 11. Le noeud 2 se positionne légèrement plus loin car il se sait en liaison directe du noeud 1 et à 1 saut des ancres 10 et 11. Son epsilon est très grand du fait de son éloignement des ancres.

On ne voit ensuite qu'un message relatif à l'ancre 15 (ligne 13) au lieu de 2

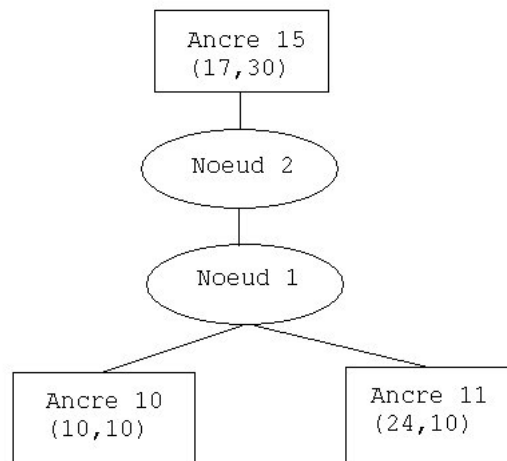


FIG. 4.1 – Topologie du test

normalement. Mais la suite nous révèle que les 2 noeuds ont reçus correctement l'information. Assitôt les noeuds se positionnent suivant le résultat définitif et émettent un message en ce sens (lignes 14 et 15) car epsilon est maintenant en dessous du seuil. La limitation de la diffusion n'est pas encore au point ce qui explique les messages suivants.

Les lignes 20 à 27 sont le résultat du passage de la sonde.

Le but des tests est de valider les simulations antérieures de nos encadrants afin de démontrer l'exactitude de leurs algorithmes de localisation.

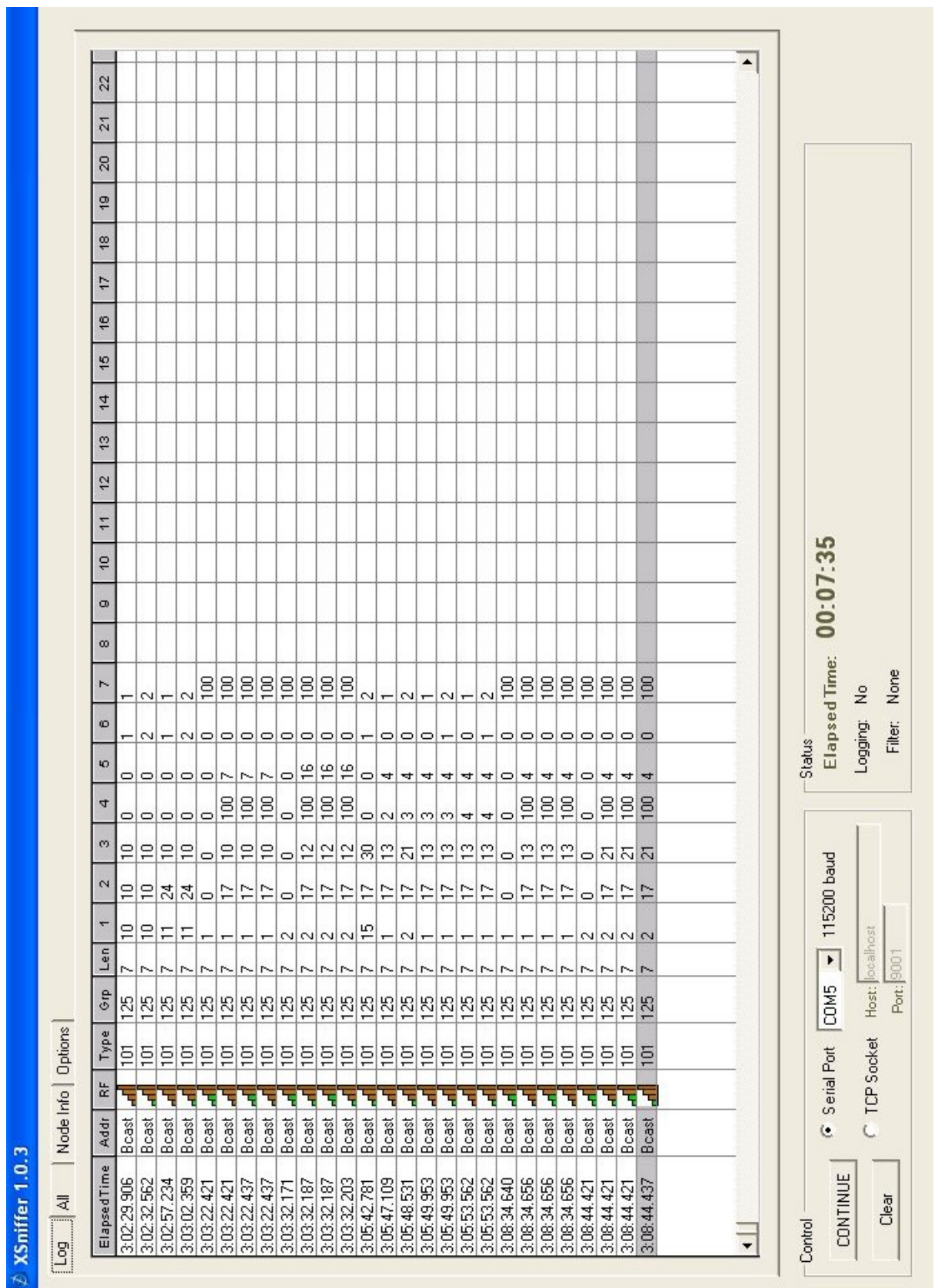


FIG. 4.2 – Résultats du test

Chapitre 5

Conclusion

Si le sujet, de prime abord, ne paraît pas très compliqué, la réalisation pratique l’a été beaucoup plus et pas seulement parce qu’il s’est agit de programmation embarquée. En effet, les outils sont dispersés et pas tous fonctionnels immédiatement. Par exemple, Crossbow fournit pour Windows un chargeur et un « sniffer », mais pas de simulateur. Sous Linux, on trouve des simulateurs mais ni chargeur pour notre plateforme matérielle, ni « sniffer ». Et les simulateurs ont des capacités en fin de compte limitées : un seul exécutable simulé à la fois, broadcast total du réseau obligatoire, etc. Ces simulateurs ne sont de plus opérationnels qu’après des manipulations pas évidentes à trouver. En bref, aucune plateforme logicielle ne permettait un travail efficace. De même, la documentation concernant les bibliothèques NesC propres au matériel est éparpillée dans les sources même des bibliothèques, sans table des matières ou autre repère pour aider à la recherche des informations. Ces problèmes ont considérablement compliqué et retardé la mise au point des programmes.

L’autre principale complication concrète concerne les émissions et réceptions de message. ZigBee utilise en couche MAC un protocole du type CSMA/CA. Malgré cela, tous les messages ne sont pas perçus. Nous n’en connaissons pas encore la raison. On suppose que soit les tampons d’entrée/sortie sont trop limités par rapport au débit des messages, soit la priorité des événements sur les tâches font que des messages ne sont pas envoyés.

Malgré tout, les premiers tests, avec un petit nombre de capteurs, sont concluants. Ils correspondent aux estimations. AT-Free n’est pas très précis mais semble assez fiable.

Nous avons découvert grâce à ce TER les réseaux de capteurs, la problématique de la localisation, le langage NesC et le système TinyOS. En résumé, beaucoup de choses.

Chapitre 6

Annexes

6.1 Installation de Moteworks (windows)

6.1.1 Compilation

Il faut avoir les fichiers suivants :

- 1 Makefile : liste les fichiers à inclure (dont le suivant) qui permettront de compiler correctement pour la cible (MICAz), et de lier le tout à TinyOS.
- 1 Makefile.component : liste les composants du programme.
- des fichiers sources en NesC (.nc)
- les fichiers de configuration et d'implémentation des modules.

La compilation se fait simplement dans un terminal Cygwin (dans le répertoire où se trouvent les fichiers sources) :

```
make micaz
```

On obtient un sous répertoire /build/micaz contenant (entre autres) :

- app.c : le source complet en C (avec TinyOS inclut).
- main.exe : fichier au format ELF ; ce n'est donc pas le véritable exécutable, mais le fichier utilisé pour écrire dans la ROM du MICAz.
- main.srec : probablement le véritable exécutable pour MICAz.

6.1.2 Chargeur : MoteConfig

Pour charger un programme dans un "mote" MICAz (Crossbow)

Prérequis :

- MoteWorks installé
- CD Wireless Sensor Network Kit
- plateforme de programmation : MIB 520 (MIB = Mote Interface Board)

Remarque préliminaire : le MIB utilise un pont USB – RS-232 (le FTDI FT2232C). Il faut installer un pilote (FTDI...) pour convertir le port USB en port COM virtuel.

Démarche (Windows XP) :

- Au 1er branchement du MIB520 sur un port USB, Windows le détecte comme un nouveau matériel
- Sélectionner "Installer à partir d'une liste ... (utilisateurs expérimentés)".

Insérer le CD-ROM. Le pilote est dans le répertoire "USB Drivers" du CD-ROM.

- Quand le pilote est installé, on doit voir apparaître 2 nouveaux ports séries (Panneau de configuration / Système / Matériel / Gestionnaire de périphériques / Ports). Noter les numéros de port assignés.
- Si les 2 ports COM virtuels sont comx et com(x+1) ; alors comx est utilisé pour programmer le MICAz tandis que com(x+1) est utilisé pour communiquer avec le MICAz.

Chargement : En utilisant MoteConfig

Connecter le MIB 520 à un port USB de l'ordinateur

Démarrer MoteConfig. Dans Settings / Interface Board, sélectionner MIB 520 et le port COM (le plus petit des deux ports créés)

Dans la fenêtre principale, sélectionner le "main.exe" à charger (ATTENTION : cette opération est nécessaire même s'il apparaît déjà correctement dans le champ textuel "Select File to be Uploaded :". En effet, dans le cas contraire, une ancienne version de l'exécutable est chargée.)

Eteindre le MICAz !

L'installer sur le MIB 520

Cliquer sur le bouton "Program"

Une fois le programme chargé, message : uploading SUCCESSFUL !

On peut aussi à partir de cette interface, lors de la programmation du capteur, affecter un numéro d'identifiant au capteur. Une autre option bien pratique (RF power) permet de modifier la puissance du signal émit par le capteur. Cette option permet de réduire jusqu'à 2 ou 3 mètres la portée d'émission des messages.

Remarque :

aussitôt après la fin du chargement, les diodes du MIB 520 reproduisent le comportement des diodes du "mote" chargé. En effet, le programme débute automatiquement, même avec le bouton OFF positionné. Ce dernier coupe l'alimentation par les piles, mais pas par le port USB (via le MIB 520).

Remarque :

MoteConfig n'est qu'une interface graphique pour le programme UISP (Micro (la lettre grecque "mu") In-System Programmer). Il est possible de compiler et charger en ligne de commande avec une instruction du type : make micaz (re)install(,n) mib520,com4

n = Id Group et Id noeud (le tout sur 16 bits) ; Id Group peut être tout sauf 0, 126 et 255 ; par défaut : 125. Voir DEFAULT_LOCAL_GROUP dans le fichier /MoteWorks/apps/MakeXbowlocal

Attention : /MoteWorks est le chemin CYGWIN. Dans la structure NTFS, cela équivaut à quelque chose comme : C :/Crossbow/cygwin/opt/MoteWorks

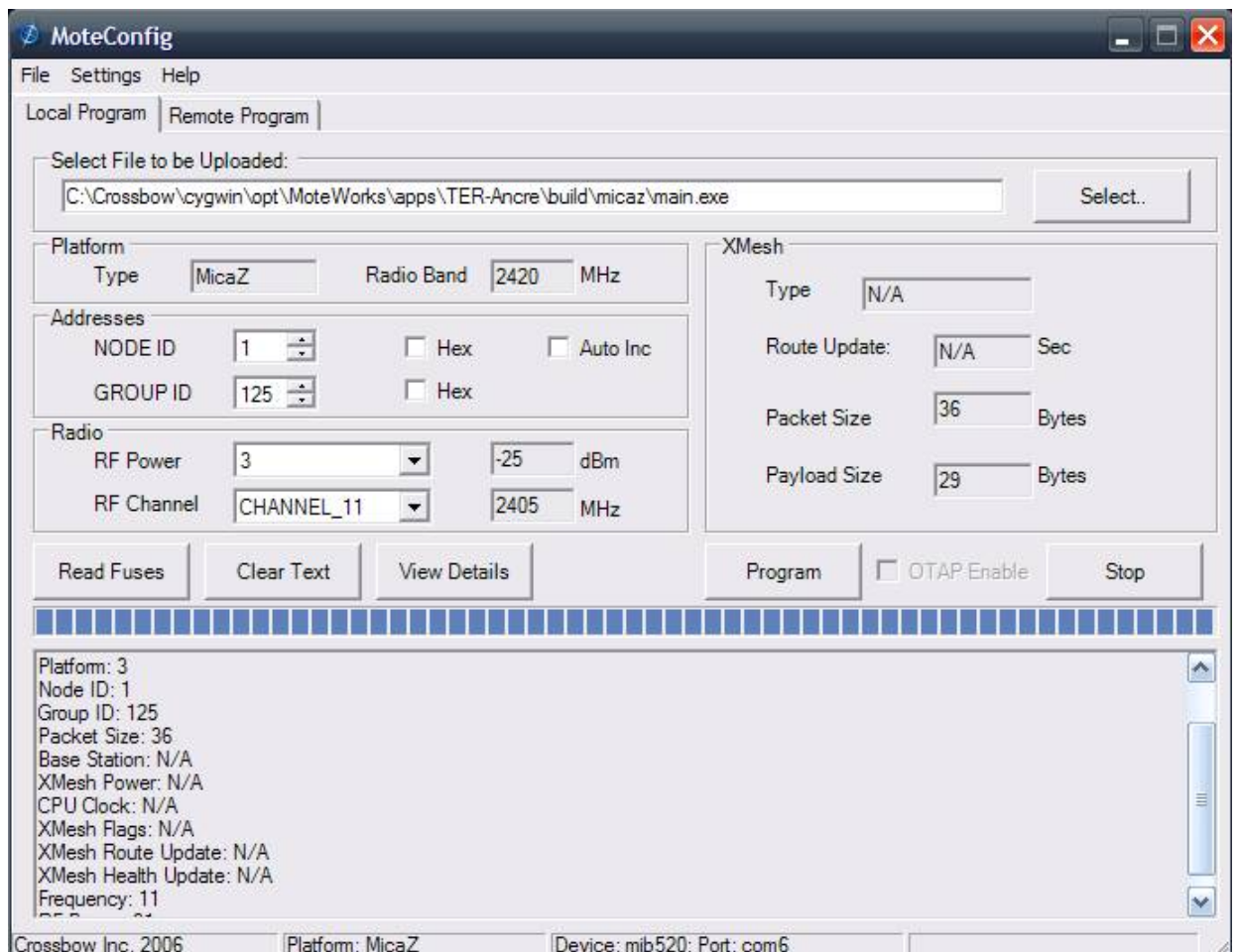


FIG. 6.1 – Le logiciel Moteconfig

6.1.3 XSniffer

Observer ("sniffer") les échanges radio entre les "motes" MICAz (Crossbow)

Prérequis :

MoteWorks installé

plateforme de programmation : MIB 520 (MIB = Mote Interface Board)

un MICAz à laisser en permanence sur le MIB 520

Remarque préliminaire : le MIB utilise un pont USB – RS-232 (le FTDI FT232RL). La communication avec le MIB se fait via 2 ports COM virtuels : comx and com(x+1) ; comx est utilisé pour programmer le MICAz tandis que com(x+1) est utilisé pour communiquer avec le MICAz.

La première opération est de compiler pour MICAz le programme qui se trouve dans : .../opt/MoteWorks/apps/general/XSniffer

Ce programme doit ensuite être chargé dans un mote (voir MoteConfig). Ce mote devra être laissé sur le MIB 520 afin que le logiciel Xsniffer puisse récupérer les échanges.

Fonctionnement :

- Démarrer Xsniffer.
- Onglet "Options" : modifier "Packet Type" (mettre General à la place de Xmesh).
- La fenêtre est alors modifiée.
- Conseil : onglet "Options" : baisser la valeur de "Measure Over Most-Recent" à 1 sec
- Choisir l'onglet "Log"
- Choisir le Serial Port com(x+1)
- Cliquer sur START

Analyse d'une entrée Log : (nom colonne : signification)

- ElapsedTime : durée écoulée depuis START
- Addr : Bcast (Broadcast)
- RF : intensité du signal radio (RSSI)
- Type : type de message (101 pour nous)
- Grp : Id group (125 par défaut)
- LEN : taille du champ DATA de la structure TOS_Msg, conformément aux structures échangées
- 1 à 64 : contenu de chaque élément du champ DATA de la structure TOS_Msg

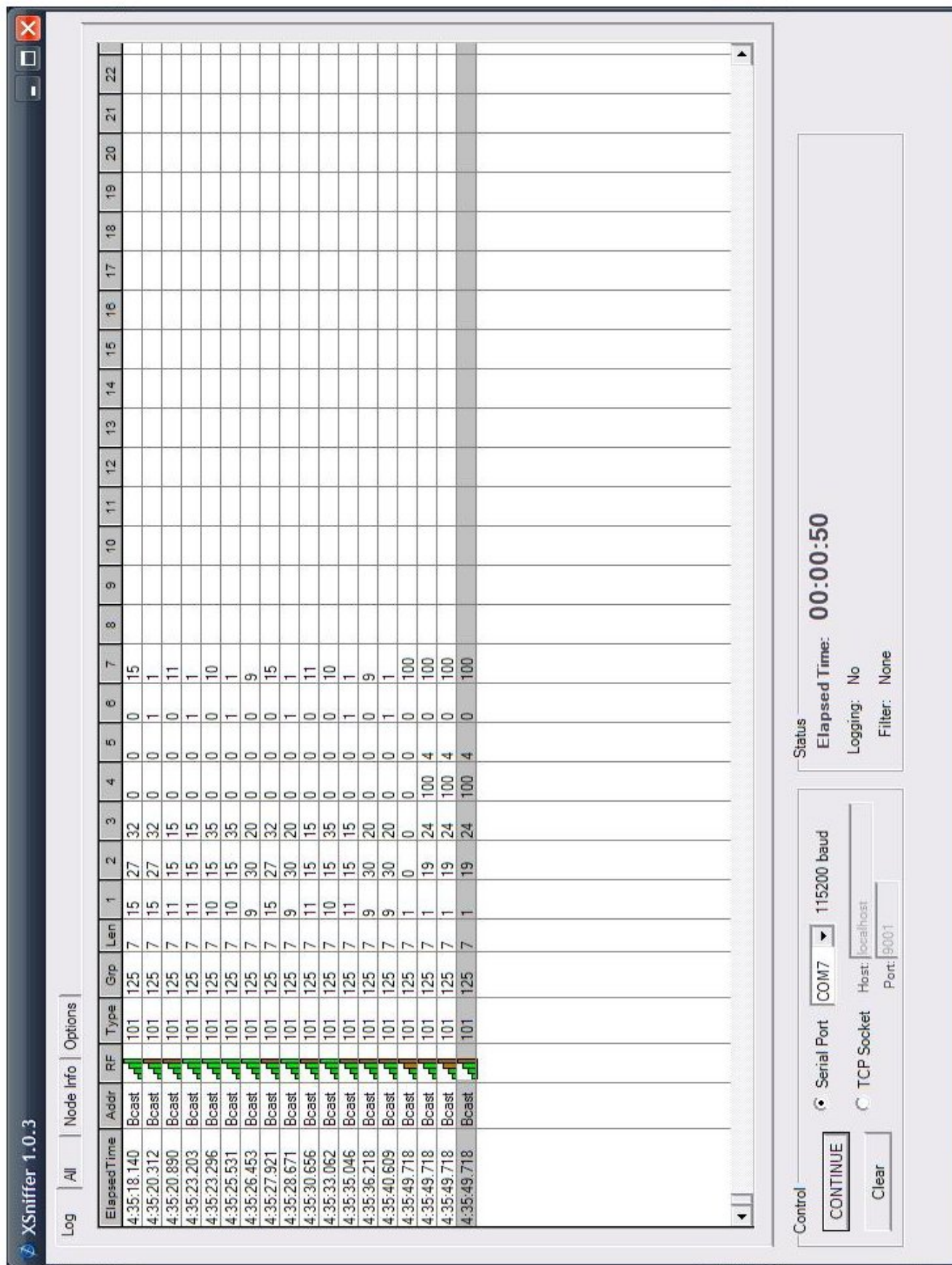


FIG. 6.2 – Le logiciel XSniffer

6.2 Intallation et configuration sous linux

Il faut tout d'abord savoir qu'il existe un live cd basé sur la distribution Ubuntu et contenant une installation pré-configurée de TinyOs1 et de TinyOs2 : XubuntOS.

iPar contre si on veut une installation sur son propre système il faut installer divers packages.

6.2.1 TinyOs 1

- Installer le JDK 1.4 de Java. Le site de tinyOs conseille d'installer le JDK d'IBM (<http://www.ibm.com/developerworks/java/jdk/>) mais tout marche très bien avec l'ancienne version du JDK toujours maintenue par sun ([j2sdk-1.4.2_17-linux-i586.bin](http://www.sun.com/javase/6/docs/bin/j2sdk-1.4.2_17-linux-i586.bin)).

Il faut maintenant installer divers .rpm (packages redhat). Si vous êtes sur une distribution basée sur Debian vous pouvez utiliser alien pour convertir les .rpm en .deb

Les fichiers se trouvent sur <http://webs.cs.berkeley.edu/tos/dist-1.1.0/tools/linux>

- avarice-2.0.20030825cvs-1.i386.rpm
- avr-binutils-2.13.2.1-1.i386.rpm
- avr-gcc-3.3tinyos-1.i386.rpm
- avr-insight-pre6.0cvs.tinyos-1.3.i386.rpm
- avr-libc-20030512cvs-1.i386.rpm
- graphviz-1.10-1.i386.rpm
- nesc-1.1-1.i386.rpm
- tinyos-tools-1.1.0-1.i386.rpm
(pour ces deux derniers : <http://webs.cs.berkeley.edu/tos/dist-1.1.0/tinyos/linux>)
- tinyos-1.1.0-1.noarch.rpm
(<http://webs.cs.berkeley.edu/tos/dist-1.1.0/tinyos/linux/tinyos-1.1.0-1.noarch.rpm>)

Le répertoire racine de tinyOs se trouve dans /opt .

Il faut modifier quelques variables globales du système :

- export TOSROOT="/opt/tinyos-1.x"
- export CLASSPATH="\$TOSROOT/tools/java/javapath :.\$TOSROOT/tools/java/net/tinyos/sim/ :/usr/local/commapi/jar/comm.jar :\$TOSROOT/tools/java/"
- export TOSDIR="\$TOSROOT/tos"
- export MAKERULES="\$TOSROOT/tools/make/Makerules"
- export PATH="/usr/local/j2sdk1.4.2_17/bin :\$PATH" (si on n'utilise pas le jdk d'ibm)

Il faut ensuite changer les permissions (chmod 666) de(s) port(s) USB que l'on va utiliser.

Un utilitaire pour tester la bonne configuration du système se trouve dans :
[/opt/tinyos-1.x/tools/scripts/toscheck](#)

Pour compiler il faut se trouver dans le répertoire du projet et taper :

```
1 make micaz
```

Ou si l'on veut construire un exécutable pour le simulateur TOSSIM :

```
1 make pc
```

on lance ensuite l'interface graphique tinyviz du simulateur avec :

```
1 /opt/tinyos-1.x/tools/java/net/tinyos/sim/tinyviz  
-run /build/pc/main.exe -5 )\
```

6.2.2 TinyOs 2

Une première façon d'installer tinyOS 2.0.2, si l'on est sur une distribution basée sur Debian, est d'utiliser le dépôt Debian de l'Université de Stanford :

```
1 deb http://tinyos.stanford.edu/tinyos/dists/  
ubuntu feisty main\
```

à rajouter dans /etc/apt/sources.list, puis faire :

```
1 apt-get update  
2 apt-get install tinyos-avr
```

Sinon la même procédure d'installation avec les paquets .rpm que pour tinyOS 1 est à réaliser, avec les versions suivantes :

- Java JDK 1.5 (Java 5).
- avr-binutils-2.15tinyos-3.i386.rpm
- avr-gcc-3.4.3-1.i386.rpm
- avr-libc-1.2.3-1.i386.rpm
- avarice-2.4-1.i386.rpm
- avr-insight-6.3-1.i386.rpm
- nesc-1.2.8a-1.i386.rpm
- tinyos-tools-1.2.4-3.i386.rpm
- tinyos-2.0.2-2.noarch.rpm

Le répertoire racine de tinyOs 2 se trouve lui aussi dans /opt .

Il faut là aussi modifier quelques variables globales du système :

- export TOSROOT="/opt/tinyos-2.x"
- export TOSDIR="\$TOSROOT/tos"
- export CLASSPATH="\$TOSROOT/support/sdk/java/tinyos.jar :."
- export MAKERULES="\$TOSROOT/support/make/Makerules"

Il n'y a plus qu'à changer le propriétaire de /opt/tinyos-2.x et modifier les permissions sur le(s) port(s) USB que l'on veut utiliser.

La branche 2.x de tinyOS est la version en développement. A nos yeux son principal défaut est qu'elle n'intègre pas tinyviz (interface graphique de simulation).

Nous avons finalement décidé de travailler avec la version 1 de tinyos plutôt que la version 2 pour deux raisons :

- le logiciel Moteworks (windows) est basé sur la version 1.
- l'interface graphique (tinyviz) du simulateur TOSSIM n'est présente que dans cette version.

6.3 Comptes rendus des réunions

Réunion 1

Compte rendu 1 : 25 / 01 / 2008

Présents :

étudiants : H. Alatrasta, S. Aliaga, K. Gouaich, J. Mathieu

enseignant(s) : J. Champ

Durant la réunion,

1. il a été convenu de faire des CR à chaque réunion.
2. un module MICAz / MIB 520 a été présenté, avec son architecture.
3. des outils pour développer sur ce module ont été proposé : XubuntOS, TinyViz et Tossim (simulateur). A ceux-ci s'ajoute MoteWorks, l'outil fournit par Xbow, le fabricant du module.
4. l'objectif du TER a été donné : implémenter sur le module des algorithmes de localisation de capteur dans un réseau (type ad-hoc) de capteurs. Les 2 algorithmes à implémenter, AT-Free et AT-Dist, ont été présenté. Dans les 2 cas, il s'agit de partir d'un graphe représentant le réseau de capteurs. Chaque nœud du graphe est un capteur et chaque arête correspond à un lien ZigBee. Certains nœuds connaissent leur position (grâce à des informations fournies par GPS). Ces nœuds portent le nom d'ancre. Les 2 algorithmes doivent permettre de connaître la position de tous les autres nœuds par une sorte de triangulation effectuée à partir des ancres. AT-Free n'utilise que la présence ou non d'arête(s) (détection ou non d'un signal ZigBee) pour trouver la position (approximative) d'un nœuds. Autrement dit, le graphe n'est pas valué. AT-Dist utilise en plus la puissance du signal (RSSI) pour valuer le graphe. La valuation est une distance déduite de RSSI.
5. un article de C. Saad et al. présentant ces algorithmes a été distribué. Un autre article, présentant un état de l'art sur les réseaux de capteurs, sera envoyé par courriel prochainement.
6. le contenu du cahier des charges a été discuté.
7. il a été convenu qu'une ébauche de cahier des charges devrait être rédigée pour la prochaine réunion.

Prochaine réunion : 01 / 02 / 2008

Réunion 2

Compte Rendu 2 : 08 / 02 / 2008

Présents :

étudiants : H. Alatrasta, S. Aliaga, J. Mathieu

enseignant(s) : Clément Saad, Julien Champ

Sujets abordés :

1. Apprendre et se familiariser avec le fonctionnement des méthodes distribuées de localisation :
 - Schéma de localisation Range-Free
 - Schéma de localisation Range-Based
2. Techniques d'approximations :
 - Technique d'approximation AT-Free
 - Technique d'approximation AT-Dist
3. Résolution de diverses questions ayant trait à l'implémentation du TER.

Commentaires :

La réunion a débuté à 14H00, nous avons reçu, en avance, un mail de Khaled Gouaich qui a justifié son absence pour cause de maladie. Nous avons commencé la réunion en discutant du cahier des charges final que nous avons remis au responsable et aux encadrants le mercredi 06 février. Nous avons ensuite abordé plusieurs points concernant la partie initiale du TER et les enseignants ont répondu à nos questions, questions essentiellement basées sur l'implémentation des algorithmes de localisation. Nous avons continué en parlant d'un processus optimal d'installation et de configuration des outils. Nous avons terminé en nous faisant une première distribution des tâches pour la semaine qui arrive. Prochain rendez-vous : vendredi prochain, ou plus tôt, selon l'avancée du projet.

Réunion 3

Compte rendu 3 : 21 / 02 / 2008

Présents :

étudiants : H. Alatrasta, S. Aliaga, K. Gouaich, J. Mathieu

enseignant(s) : Jean-Claude König, Clément Saad, Julien Champ

Sujets abordés :

1. mettre au point une installation complète et opérationnelle de l'outil de programmation afin d'implémenter le code NesC.
2. réfléchir sur l'algorithme de diffusion et élaborer une méthode pour l'algorithme de détection des nœuds "voisins".

Commentaires : Le sujet de la réunion était, essentiellement, porté sur l'état de notre avancement par rapport aux différentes tâches qui nous ont été attribuées lors des réunions précédentes. Une planification et distribution de nouvelles fonctions ont été accordées par les encadrants concernant les algorithmes ainsi que l'installation des outils de programmation. Il a été convenu aussi que des essais sur les modules seront peut être envisageables si les circonstances le permettent, la difficulté repose essentiellement sur l'adaptation de la plate forme de programmation.

Réunion 4

Nous avons réussi à émettre un message mais nous n'arrivons pas à en remplir les champs de données.

Réunion 5

Nous sommes arrivés transmettre, recevoir et traiter des données. Nous arrivons aussi à relayer des messages.

Notre encadrant C. Saad nous a expliqué quelques subtilités des seuils dans AT-Free que nous avions oublié.

6.4 Fiche technique des capteurs micaz

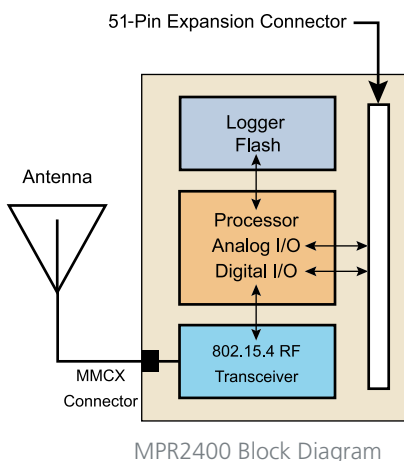
MICAz

WIRELESS MEASUREMENT SYSTEM

- 2.4 GHz IEEE 802.15.4, Tiny Wireless Measurement System
- Designed Specifically for Deeply Embedded Sensor Networks
- 250 kbps, High Data Rate Radio
- Wireless Communications with Every Node as Router Capability
- Expansion Connector for Light, Temperature, RH, Barometric Pressure, Acceleration/Seismic, Acoustic, Magnetic and other Crossbow Sensor Boards

Applications

- Indoor Building Monitoring and Security
- Acoustic, Video, Vibration and Other High Speed Sensor Data
- Large Scale Sensor Networks (1000+ Points)



MICAz

The MICAz is a 2.4 GHz Mote module used for enabling low-power, wireless sensor networks.

Product features include:

- IEEE 802.15.4 compliant RF transceiver
- 2.4 to 2.48 GHz, a globally compatible ISM band
- Direct sequence spread spectrum radio which is resistant to RF interference and provides inherent data security
- 250 kbps data rate
- Supported by MoteWorks™ wireless sensor network platform for reliable, ad-hoc mesh networking
- Plug and play with Crossbow's sensor boards, data acquisition boards, gateways, and software

MoteWorks™ enables the development of custom sensor applications and is specifically optimized for low-power, battery-operated networks. MoteWorks is based on the open-source TinyOS operating system and provides reliable, ad-hoc mesh networking, over-the-air-programming capabilities, cross development tools, server middleware for enterprise network integration and client user interface for analysis and a configuration.

Processor & Radio Platform (MPR2400CA)

The MPR2400 is based on the Atmel ATmega128L. The ATmega128L is a low-power microcontroller which runs MoteWorks from its internal flash memory. A single processor board (MPR2400) can be configured to run your sensor application/processing and the network/radio communications stack simultaneously. The 51-pin expansion connector supports Analog Inputs, Digital I/O, I2C, SPI and UART interfaces. These interfaces make it easy to connect to a wide variety of external peripherals. The MICAz (MPR2400) IEEE 802.15.4 radio offers both high speed (250 kbps) and hardware security (AES-128).

Sensor Boards

Crossbow offers a variety of sensor and data acquisition boards for the MICAz Mote. All of these boards connect to the MICAz via the standard 51-pin expansion connector. Custom sensor and data acquisition boards are also available. Please contact Crossbow for additional information.

Processor/Radio Board	MPR2400CA	Remarks
Processor Performance		
Program Flash Memory	128K bytes	
Measurement (Serial) Flash	512K bytes	> 100,000 Measurements
Configuration EEPROM	4K bytes	
Serial Communications	UART	0-3V transmission levels
Analog to Digital Converter	10 bit ADC	8 channel, 0-3V input
Other Interfaces	Digital I/O,I2C,SPI	
Current Draw	8 mA	Active mode
	< 15 μ A	Sleep mode
RF Transceiver		
Frequency band ¹	2400 MHz to 2483.5 MHz	ISM band, programmable in 1 MHz steps
Transmit (TX) data rate	250 kbps	
RF power	-24 dBm to 0 dBm	
Receive Sensitivity	-90 dBm (min), -94 dBm (typ)	
Adjacent channel rejection	47 dB	+ 5 MHz channel spacing
	38 dB	- 5 MHz channel spacing
Outdoor Range	75 m to 100 m	1/2 wave dipole antenna, LOS
Indoor Range	20 m to 30 m	1/2 wave dipole antenna
Current Draw	19.7 mA	Receive mode
	11 mA	TX, -10 dBm
	14 mA	TX, -5 dBm
	17.4 mA	TX, 0 dBm
	20 μ A	Idle mode, voltage regular on
	1 μ A	Sleep mode, voltage regulator off
Electromechanical		
Battery	2X AA batteries	Attached pack
External Power	2.7 V - 3.3 V	Molex connector provided
User Interface	3 LEDs	Red, green and yellow
Size (in)	2.25 x 1.25 x 0.25	Excluding battery pack
(mm)	58 x 32 x 7	Excluding battery pack
Weight (oz)	0.7	Excluding batteries
(grams)	18	Excluding batteries
Expansion Connector	51-pin	All major I/O signals

Notes

¹5 MHz steps for compliance with IEEE 802.15.4/D18-2003.

Specifications subject to change without notice



MIB520CB Mote Interface Board

Base Stations

A base station allows the aggregation of sensor network data onto a PC or other computer platform. Any MICAz Mote can function as a base station when it is connected to a standard PC interface or gateway board. The MIB510 or MIB520 provides a serial/USB interface for both programming and data communications. Crossbow also offers a stand-alone gateway solution, the MIB600 for TCP/IP-based Ethernet networks.

Ordering Information

Model	Description
MPR2400CA	2.4 GHz MICAz Processor/Radio Board
WSN-START2400CA	2.4 GHz MICAz Starter Kit
WSN-PRO2400CA	2.4 GHz MICAz Professional Kit

Document Part Number: 6020-0060-04 Rev A

Chapitre 7

Glossaire

- Broadcast : Dans le domaine des télécommunications en général, le Broadcasting désigne une méthode de diffusion de données à partir d'une source unique vers un ensemble de récepteurs. Dans le domaine de l'informatique le Broadcast désigne une méthode de transmission de données à l'ensemble des machines d'un réseau. Broadcast ou Diffusion est aussi une mode d'émission de données à destination de plusieurs utilisateurs. Utilise une adresse spécifique de diffusion et un protocole en mode "non connecté".
- Capteur : Un capteur est un dispositif transformant l'état d'une grandeur physique observée en une grandeur utilisable exemple : une tension électrique, une hauteur de mercure, une intensité, la déviation d'une aiguille etc. Pour nous la grandeur physique observée est transparente.
- Connection : Lien entre deux entités communicantes d'un même niveau de protocole.
- Node (noeud) : Désigne un dispositif comportant plusieurs voies radio et réalisant de la commutation de trames ou de paquets.
- Sans fil : Dispositif ou système servant à une application de télécommunication ou à plusieurs sans que des fils soient nécessaires pour communiquer entre les nœuds, lesquels utilisent habituellement et plutôt des radiofréquences.
- Topologie : La topologie est la notion qui caractérise le positionnement des éléments d'un ensemble les uns par rapport aux autres. On parlera donc fréquemment de la topologie d'un circuit imprimé ou d'un circuit intégré (dans ce cas ce sont les positions respectives des microcomposants élémentaires qui sont en jeu) ou aussi d'une topologie de réseau.
- ZigBee : ZigBee est un protocole de haut niveau permettant la communication de petites radios, à consommation réduite, basé sur le standard IEEE 802.15.4 pour les réseaux à dimension personnelle (Wireless Personal Area Networks : WPANs). Cette technologie a pour but la com-

munication de courte distance telle que le propose déjà la technologie Bluetooth, tout en étant moins chère et plus simple.

Chapitre 8

Bibliographie

1. AT-Family : Distributed Methods for Localization in Sensor Networks (Clément Saad, Abderrahim Benslimane, Jean-Claude König).
2. Wireless sensor networks : a survey (I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci).
3. documentation du matériel : www.xbow.com
4. documentation sur TinyOS : <http://fr.wikipedia.org/wiki/TinyOS>
5. documentation sur le langage nesC : <http://nescc.sourceforge.net/>
6. notre site web pour le projet : <http://hugo.alatristasalas.free.fr/>